

ANALYZING THE LENGTH-BASED ATTACK ON POLYCYCLIC GROUPS

DAVID GARBER, DELARAM KAHROBAEI, HA T. LAM

ABSTRACT. After the Anshel-Anshel-Goldfeld (AAG) key-exchange protocol was introduced in 1999, it was implemented and studied with braid groups and with the Thompson group as its underlying platforms. The length-based attack, introduced by Hughes and Tannenbaum, has been used to extensively study AAG with the braid group as the underlying platform. Meanwhile, a new platform, using polycyclic groups, was proposed by Eick and Kahrobaei.

In this paper, we show that with a high enough Hirsch length, the polycyclic group as an underlying platform for AAG is resistant to the length-based attack. In particular, polycyclic groups could provide a secure platform for any cryptosystem based on conjugacy search problem such as non-commutative Diffie-Hellman, ElGamal and Cramer-Shoup key exchange protocols.

1. INTRODUCTION

The Anshel-Anshel-Goldfeld (AAG) key-exchange protocol was introduced in 1999 [1]. Following its introduction, AAG was extensively studied using different groups as its underlying platform. Ko et al. used braid groups [12], while Shpilrain and Ushakov [16] proposed Thompson's group.

Hughes and Tannenbaum [9] originated the length-based attack (LBA) on AAG with its implementation in braid groups. They emphasized the importance of choosing the correct length function. Later, Garber et al. [6] gave several realizations of this approach, particularly a length function for the braid group and experimental results suggesting that the attack is infeasible given the parameters in existing protocols. However, Garber et al. [5] also suggested an extension of the length-based attack which uses memory which succeeded in breaking AAG. Similar attack was implemented against a system based on the Thompson group [15]. Most recently, Myasnikov and Ushakov [14] analyzed reasons behind the failure of the previous implementations of the LBA, e.g. the occurrence of *commutator-type peak*, and gave an experimental evidence that the LBA can be modified to break AAG with a high rate of success. However, this work is again done with respect to braid groups.

Meanwhile, a different platform for AAG, that of the *polycyclic group*, is suggested by Eick and Kahrobaei [3]. In polycyclic groups, the word problem can be solved efficiently, but known solutions to the conjugacy problem are far less efficient. Using experimental results, Eick and Kahrobaei showed that while the conjugacy problem can be solved within seconds using polycyclic groups with small Hirsch length, the conjugacy problem in polycyclic groups with high Hirsch length can not be solved even in a much longer time.

Delaram Kahrobaei is partially supported by the Office of Naval Research grant N000141210758 and also supported by PSC-CUNY grant from the CUNY research foundation, as well as the City Tech foundation.

Taking inspiration from this result, we investigate the success rate of the length-based attack on polycyclic groups, especially those with high Hirsch length. Toward this end, we first construct polycyclic groups of high Hirsch length using a method introduced by Holt et al. [8]. Then, we implement the LBA using the algorithms presented in [5, 6, 14]. The experimental results that we collect suggest that with high enough Hirsch length, the polycyclic group as a platform for AAG is resistant to the known different variants of the LBA.

As a wider application, we note that the conjugacy search problem is the basis for various cryptographic protocols besides AAG, such as the non-commutative Diffie-Hellman key exchange [12], the non-commutative El-Gamal key exchange [10], the non-abelian Cramer-Shoup key exchange [2] and the non-commutative digital signatures [11]. The LBA can be applied to all of these protocols; therefore, a secured platform group, proven resistant to known variants of the LBA, such as polycyclic groups, can help strengthen them.

The paper is organized as follows. In Section 2, we introduce the Anshel-Anshel-Goldfeld key exchange protocol. In Section 3, we give a short review of polycyclic groups and the construction that we have used. In Section 4, we review the length-based attack, and in Section 5, we present the experiments, results and conclusions that we have made.

2. THE ANSHEL-ANSHEL-GOLDFELD KEY EXCHANGE PROTOCOL

Following [14], we introduce the Anshel-Anshel-Goldfeld key-exchange protocol. For more details, see [1]. As usual, we use two entities, called Alice and Bob, for presenting the two parties which plan to communicate over an insecure channel.

Let G be a group with generators g_1, \dots, g_n . First, Alice chooses, as her public set, $\bar{a} = (a_1, \dots, a_{N_1})$ where $a_i \in G$ and Bob chooses, as his public set, $\bar{b} = (b_1, \dots, b_{N_2})$ where $b_i \in G$. They both publish their sets. Alice then chooses her private key $A = a_{s_1}^{\varepsilon_1} \dots a_{s_L}^{\varepsilon_L}$ where $a_{s_i} \in \bar{a}$ and $\varepsilon_i \in \{\pm 1\}$. Bob also chooses his private key $B = b_{t_1}^{\delta_1} \dots b_{t_L}^{\delta_L}$ where $b_{t_i} \in \bar{b}$ and $\delta_i \in \{\pm 1\}$. Alice computes $b'_i = A^{-1}b_iA$ for all $b_i \in \bar{b}$ and sends it to Bob. Bob also computes $a'_i = B^{-1}a_iB$ for all $a_i \in \bar{a}$ and sends it to Alice. Now, the shared secret key is $K = A^{-1}B^{-1}AB$. Alice can compute this key by

$$\begin{aligned} K_A &= A^{-1}a_{s_1}'^{\varepsilon_1} \dots a_{s_L}'^{\varepsilon_L} = A^{-1}(B^{-1}a_{s_1}B)^{\varepsilon_1} \dots (B^{-1}a_{s_L}B)^{\varepsilon_L} \\ &= A^{-1}B^{-1}a_{s_1}^{\varepsilon_1} \dots a_{s_L}^{\varepsilon_L}B = A^{-1}B^{-1}AB = K \end{aligned}$$

Bob can likewise compute $K_B = B^{-1}b_{t_1}'^{\delta_1} \dots b_{t_L}'^{\delta_L} = B^{-1}A^{-1}BA$, then the shared key is $K = K_B^{-1}$.

In order to find K , the ear-dropper needs to find either $A' \in \langle a_1, \dots, a_{N_1} \rangle$ such that $\bar{b}' = A'^{-1}\bar{b}A'$ or find $B' \in \langle b_1, \dots, b_{N_2} \rangle$ such that $\bar{a}' = B'^{-1}\bar{a}B'$. Thus, the security of AAG is based on the assumption that the subgroup-restricted simultaneous conjugacy search problem is hard.

3. POLYCYCLIC GROUPS

In this section, we give a short review of the polycyclic presentation and discuss how we generate polycyclic groups of high Hirsch length. For more details, see [8].

3.1. The polycyclic presentation. Recall that G is a *polycyclic group* if it has a polycyclic series, i.e., a subnormal series $G = G_1 \triangleright G_2 \triangleright \cdots \triangleright G_{n+1} = \{1\}$ with non-trivial cyclic factors. The polycyclic generating sequence of G is the n -tuple (g_1, \dots, g_n) such that $G_i = \langle g_i, G_{i+1} \rangle$ for $1 \leq i \leq n$.

Every polycyclic group has a finite presentation of the form:

$$\langle g_1, \dots, g_n \mid g_j^{g_i} = w_{ij}, g_j^{g_i^{-1}} = v_{ij}, g_k^{r_k} = u_{kk} \text{ for } 1 \leq i < j \leq n \text{ and } k \in I \rangle$$

where w_{ij}, v_{ij}, u_{kk} are words in the generators g_{i+1}, \dots, g_n and I is the set of indices $i \in \{1, \dots, n\}$ such that $r_i = [G_i : G_{i+1}]$ is finite. Here a^b stands for $b^{-1}ab$.

Using induction, we see that each element of G defined by this presentation can be uniquely written as $g = g_1^{e_1} \cdots g_n^{e_n}$ with $e_i \in \mathbb{Z}$ for $1 \leq i \leq n$, and $0 \leq e_i < r_i$ for $i \in I$. This is the *normal form* of an element. If every element in the group can be uniquely presented in the normal form, then the polycyclic presentation is called *consistent*. Note that every polycyclic group has a consistent polycyclic presentation.

The *Hirsch length* of a polycyclic group is the number of i such that $r_i = [G_i : G_{i+1}]$ is infinite. This number is invariant of the chosen polycyclic sequence.

3.2. Polycyclic groups for AAG. Polycyclic groups are suitable as platform groups for AAG for several reasons. First, the word problem can be solved efficiently using the collection algorithm [3]. Second, the search conjugacy problem has no efficient solution in general polycyclic groups. This assessment is due to Eick and Kahrobaei [3], using the following experiment: let $K = \mathbb{Q}[x]/(f_w)$ be an algebraic number field for a cyclotomic polynomial f_w , where w is a primitive r -th root of unity, and let $G(w) = O \rtimes U$ where O is the maximal order and U the unit group of K , r the order of w and $h(G(w))$ the Hirsch length. The average time used for 100 applications of the collection algorithm on random words and the average time used for 100 applications of the conjugacy algorithm on random conjugates are:

r	h(G(w))	Collection	Conjugation
3	2	0.00 sec	9.96 sec
4	2	0.00 sec	9.37 sec
7	6	0.01 sec	10.16 sec
11	14	0.05 sec	> 100 hours

We can see that the collection algorithm works very fast even for polycyclic groups of high Hirsch length, which enables the word problem to be solved efficiently; but the solution to the conjugacy problem is not efficient for polycyclic groups having high Hirsch length.

4. THE LENGTH-BASED ATTACK

The length-based attack is a probabilistic attack against the conjugacy search problem in general, and against AAG in principal, with the goal of finding Alice's (or Bob's) private key. It is based on the idea that a conjugation of the correct element should decrease the length of the captured package. Using the notations of Section 2, the captured package is $\bar{b}' = (b'_1, \dots, b'_{N_2})$ where $b'_i = A^{-1}b_iA$. If we conjugate \bar{b}' with elements from the group $\langle a_1, \dots, a_{N_1} \rangle$ and the length of the resulting tuple has been decreased, then we know that we have found a conjugating factor. The process of conjugation is then repeated with the decreased length tuple

until another conjugating factor is found. The process ends when the conjugated captured package is the same as $\bar{b} = (b_1, \dots, b_{N_2})$, which is known. Then, the conjugate can be recovered by reversing the sequence of conjugating factors. For more details on the length-based attack, see [13, 14].

4.1. Variants of the LBA. In [5, 6, 14, 15], several variants of the LBA are presented. Here, we give four variants of LBA that we implemented against AAG based on the polycyclic group. In all these algorithms, the following input and output are expected:

- INPUT: $\bar{a} = (a_1, \dots, a_{N_1})$, $\bar{b} = (b_1, \dots, b_{N_2})$ and $\bar{b}' = (b'_1, \dots, b'_{N_2})$, such that $b'_i = b_i^A$ for $i = 1, \dots, N_2$
- OUTPUT: An element $A' \in \langle a_1, \dots, a_{N_1} \rangle$ such that $b'_i = b_i^{A'}$ for $i = 1, \dots, N_2$ or FAIL if the algorithm cannot find such A'

The following notation is also used: if $\bar{c} = (c_1, \dots, c_k)$, then its *total length* $|\bar{c}|$ is $\sum_{i=1}^k |c_i|$.

4.1.1. LBA with backtracking. The most straight-forward variant of LBA (Algorithm 1) conjugates \bar{b}' directly with $a_i^{\pm 1} \in \{a_1, \dots, a_{N_1}\}$. This is termed “LBA with backtracking” by Myasnikov and Ushakov [14].

Algorithm 1 LBA with backtracking

```

1: Initialize  $S = \{(\bar{b}', \text{id}_G)\}$ .
2: while  $S \neq \emptyset$  do
3:   Choose  $(\bar{c}, x) \in S$  such that  $|\bar{c}|$  is minimal. Remove  $(\bar{c}, x)$ 
4:   for  $i = 1, \dots, N_1$  and  $\varepsilon = \pm 1$  do
5:     Compute  $\delta_{i,\varepsilon} = |\bar{c}| - |\bar{c}^{a_i^\varepsilon}|$ 
6:     if  $\bar{c}^{a_i^\varepsilon} = \bar{b}'$  then output inverse of  $xa_i^\varepsilon$  and stop
7:     if  $\delta_{i,\varepsilon} > 0$  then ▷ length has been decreased
8:       Add  $(\bar{c}^{a_i^\varepsilon}, xa_i^\varepsilon)$  to  $S$ 
9:     end if
10:  end for
11: end while
12: Otherwise, output FAIL ▷ no more element to conjugate

```

4.1.2. LBA with a dynamic set. Through analysis, Myasnikov and Ushakov concluded that different types of peaks make LBA unsuccessful [14]. To overcome this, they suggested a new version of the algorithm, which they termed “LBA with a dynamic set”. Here, depending on whether an a_i causes a length reduction, either only the conjugates and products involving a_i are added to the dynamic set, or, in the unlucky case of no length reduction, all conjugates and two generators products are added. Their experimental results suggest that this algorithm works especially well in the case of keys constructed from long generators, but not worse than the naive algorithm in other cases. Algorithm 2 is a modified version of their algorithm, which we implemented to attack AAG based on the polycyclic group.

Algorithm 2 LBA with a dynamic set

```

1: Initialize  $S = \{(\bar{b}', \text{id}_G)\}$ .
2: while  $S \neq \emptyset$  do
3:   Choose  $(\bar{c}, x) \in S$  such that  $|\bar{c}|$  is minimal. Remove  $(\bar{c}, x)$ 
4:   for  $i = 1, \dots, N_1$  and  $\varepsilon = \pm 1$  do
5:     Compute  $\delta_{i,\varepsilon} = |\bar{c}| - |\bar{c}^{a_i^\varepsilon}|$ 
6:   end for
7:   if  $\delta_{i,\varepsilon} \leq 0$  for all  $i$  then
8:     Define  $\bar{a}_{\text{ext}} = \bar{a} \cup \{x_i x_j x_i^{-1}, x_i x_j, x_i^2 \mid x_i, x_j \in \bar{a}^{\pm 1}, i \neq j\}$ 
9:   else Define  $\bar{a}_{\text{ext}} = \bar{a} \cup \{x_j x_m x_j^{-1}, x_m x_j, x_j x_m, x_m^2 \mid x_j \in \bar{a}^{\pm 1}, m \neq j\}$ 
      where  $x_m$  such that  $\delta_m = \max\{\delta_{i,\varepsilon} \mid i = 1, \dots, N_1\}$ 
10:  end if
11:  for all  $w \in \bar{a}_{\text{ext}}$  do
12:    Compute  $\delta_w = |\bar{c}| - |\bar{c}^w|$ 
13:  end for
14:  if  $\bar{c}^w = \bar{b}$  then output inverse of  $xw$  and stop
15:  if  $\delta_w > 0$  then ▷ length has been decreased
16:    Add  $(\bar{c}^w, xw)$  to  $S$ 
17:  end if
18: end while
19: Otherwise, output FAIL ▷ no more element to conjugate

```

4.1.3. *LBA with memory 1.* Based on [5], we present another variant of LBA that is based on a fixed-size memory allocated for the algorithm. Here, S holds M tuples every round and is sorted by the first element (with respect to the length of conjugated element) of each tuple. In every round, the smallest element of S is removed and conjugated by all the generators and their inverses, the conjugated tuples are added back into S depending on whether there is still a free place in S . If there is no more places in S , and if the conjugated tuple is smaller than the largest element in S , swap them, and then re-sort S . Note that since S is always kept sorted, any operation to find the “smallest element” costs constant time. We use a time-out that can be defined as the halting condition.

4.1.4. *LBA with memory 2.* A different algorithm, truer to the spirit of [5], is also considered. In this algorithm again, S holds M tuples every round. In every round, all elements of S are conjugated, but only the M smallest conjugated tuples (with respect to their length) are added back into S . Here again, for the halting condition, we use a time-out that is defined by the user (as in the previous variant).

4.2. **The length function.** In the implementation of LBA, the choice of the length function is important (see [5, 7]). In our case, the length of a word is chosen to be the sum of the absolute values of the exponents in its normal form. We choose this function because the experimental results presented below show that it satisfies the requirement $\ell(a^{-1}ba) \gg \ell(b)$ (as needed for a length function associated to LBA).

The experiments are done by first constructing a polycyclic group G of Hirsch length $h(G)$ following the construction in Section 5.1 below. Then, an element b of length between 10 and 13 is randomly chosen; we choose elements of this length for consistency in the length-based attack parameters. Another random element a

Algorithm 3 LBA with Memory 1

```

1: Initialize  $S = \{(|\bar{b}'|, \bar{b}', \text{id}_G)\}$ .
2: while not time-out do
3:   Choose  $(|\bar{c}|, \bar{c}, x) \in S$  such that  $|\bar{c}|$  is minimal. Remove  $(|\bar{c}|, \bar{c}, x)$ 
4:   for  $i = 1, \dots, N_1$  and  $\varepsilon = \pm 1$  do
5:     Compute  $\bar{c}^{a_i^\varepsilon}$ 
6:     if  $\bar{c}^{a_i^\varepsilon} = \bar{b}$  then output inverse of  $xa_i^\varepsilon$  and stop
7:     if  $\text{Size}(S) < M$  then
8:       Add  $(|\bar{c}^{a_i^\varepsilon}|, \bar{c}^{a_i^\varepsilon}, xa_i^\varepsilon)$  to  $S$  and sort  $S$  by first element of every tuple
9:     else ▷ no more space in S
10:      if  $|\bar{c}^{a_i^\varepsilon}|$  is smaller than first element of all tuples in  $S$  then swap them
11:    end if
12:  end for
13: end while
14: Otherwise, output FAIL ▷ no more element to conjugate

```

Algorithm 4 LBA with Memory 2

```

1: Initialize  $S = \{(|\bar{b}'|, \bar{b}', \text{id}_G)\}$ .
2: while not time-out do
3:   for  $(|\bar{c}|, \bar{c}, x) \in S$  do
4:     Remove  $(|\bar{c}|, \bar{c}, x)$  from  $S$ 
5:     Compute  $\bar{c}^{a_i^\varepsilon}$  for all  $i \in \{1 \dots N_1\}$  and  $\varepsilon = \pm 1$ 
6:     if  $\bar{c}^{a_i^\varepsilon} = \bar{b}$  then output inverse of  $xa_i^\varepsilon$  and stop
7:     Save  $(|\bar{c}^{a_i^\varepsilon}|, \bar{c}^{a_i^\varepsilon}, xa_i^\varepsilon)$  in  $S'$ 
8:   end for
9:   After finished all conjugations, sort  $S'$  by the first element of every tuple
10:  Copy the smallest  $M$  elements into  $S$  and delete the rest of  $S'$ 
11: end while
12: Otherwise, output FAIL

```

of the same length interval is chosen and b^a is computed, and finally, we compute $|b^a| - |b|$. We ran 100 tests for each group and the average difference is recorded.

Polynomial	$h(G)$	Average difference
$x^2 - x - 1$	3	79.92
$x^5 - x^3 - 1$	7	80.17
$x^{11} - x^3 - 1$	16	44.93

As we can see, the average difference is large, in particular, $|b^a| - |b|$ is significantly larger than $|a|$, indicating that the condition $\ell(a^{-1}ba) \gg \ell(b)$ is indeed satisfied.

5. EXPERIMENTAL RESULTS

Our goal is to test the feasibility of the LBA on AAG based on the polycyclic group. To that end, we implemented the four variants of the LBA presented in Section 4 and ran experiments on polycyclic groups with different Hirsch lengths.

5.1. Implementation details. Each polycyclic group is generated by choosing an irreducible polynomial f over $\mathbb{Z}[x]$, then f defines an algebraic field F over \mathbb{Q} . Let O_F be its maximal order and U_F its unit group, then $O_F \rtimes U_F$ is the desired polycyclic group. This construction follows [8] and is a part of the Polycyclic Package of GAP [4].

A random element a_i , for Alice's public set, or b_i , for Bob's public set, is generated by taking either some random generators of the group or their inverses and multiplying them together, while maintaining that the length of the element is between a predefined minimum and maximum. By this method, we have more control over the length of the element.

Alice's private key A is generated by taking a fixed number of random elements in $\bar{a} = (a_1, \dots, a_{N_1})$ and multiplying them together. Here we forgo control over length to preserve interesting cases of conjugations actually decreasing the length of b_i , i.e. a commutator-type peak.

5.2. Results. We performed several sets of tests, all of which were run on an Intel Core I7 quad-core 2.0GHz computer with 12GB of RAM, running Ubuntu version 12.04 with GAP version 4.5 and 10GB of memory allowance. In all these tests, the polycyclic group G having Hirsch length $h(G)$ is constructed by the above method with polynomial f . The size of Alice's and Bob's public sets are both $N_1 = N_2 = 20$.

5.2.1. The effect of the Hirsch length. In the first set of tests, the length of each random element a_i or b_i is in the interval $[L_1, L_2] = [10, 13]$ and Alice's private key is the product of $L = 5$ random elements in Alice's public set. The time for each batch of 100 tests are recorded together with its success rate. In each case, a time-out of 30 minutes is enforced for each test. The following results are obtained by Algorithm 2.

Polynomial	$h(G)$	Time	Success rate
$x^2 - x - 1$	3	0.20 hours	100%
$x^5 - x^3 - 1$	7	76.87 hours	35%
$x^7 - x^3 - 1$	10	94.43 hours	8%
$x^9 - 7x^3 - 1$	14	95.18 hours	5%
$x^{11} - x^3 - 1$	16	95.05 hours	5%

From this table, we can see that with a small Hirsch length, the length-based attack breaks AAG easily with high success rate. However, as the Hirsch length is increased to 7, the success rate decreases. The tipping point is Hirsch length 10, where the success rate is only 8%. At polycyclic groups with higher Hirsch lengths, we can see the effect of the time-out more prominently as the total time did not increase much more, but the success rate is dropped to only 5%.

5.2.2. The effect of the key length. In the second set of tests, we vary the number of elements L that compose Alice's private key. Myasnikov and Ushakov [14] suggested that the LBA with a dynamic set (Algorithm 2) has a high success rate with long generators, i.e. random elements have longer length $[L_1, L_2]$. Therefore, we also vary the length of random elements according to the parameters in [14]. The following results are obtained by Algorithm 2, also with a time-out of 30 minutes.

Polynomial	$h(G)$	[10,13]	[20,23]		[40,43]
		$L = 10$	$L = 10$	$L = 20$	$L = 50$
$x^7 - x^3 - 1$	10	2%	0%	0%	0%
$x^9 - 7x^3 - 1$	14	0%	0%	0%	0%
$x^{11} - 3x^3 - 1$	17	0%	0%	0%	0%

The results of this set of tests indicate that just by increasing the number of generators of Alice's private key from 5 (as in the previous set of tests) to 10, the LBA already fails with polycyclic groups having Hirsch length as small as 10.

5.2.3. Comparing the four variants of the LBA. In this paper, we compare the success rate of the four variants of the LBA *for the first time* on any platform. For comparing the success rate of the four variants of the LBA, we purposely choose the test parameters very small in this set of tests. They are as follows: $N_1 = N_2 = 20$, $[L_1, L_2] = [5, 8]$, $L = 5$, there is a time-out of 30 minutes and a memory of size $M = 500$. The polynomial used is $f = x^3 - x - 1$, generating a polycyclic group of Hirsch length 4.

Algorithm	Time	Success rate
Algorithm 1 (LBA with backtracking)	0.57 hours	58%
Algorithm 2 (LBA with a dynamic set)	37.35 hours	95%
Algorithm 3 (Memory 1)	32.00 hours	36%
Algorithm 4 (Memory 2)	4.01 hours	92%

Algorithm 2 gives the best success rate but took much longer than Algorithm 4 which gives a similar success rate in much shorter time. We conclude that with a sufficient size of memory, Algorithm 4 is the best variant of the LBA.

5.2.4. Using the four variants of the LBA on our test parameters. In the fourth set of tests, we want to see the effect of the four different variants of the LBA presented in Section 4.1 applied to our test parameters. Therefore, we keep the same following parameters for all the algorithms: the length of each random element is in the interval $[L_1, L_2] = [10, 13]$, Alice's private key is the product of 10 elements and the length of both public sets are $N_1 = N_2 = 20$. There is a time-out of 30 minutes per test and in the case of the two memory variants of the LBA, Algorithm 3 and Algorithm 4, a memory of size $M = 1000$ is used. The same polycyclic group G having Hirsch length 14 constructed from the polynomial $x^9 - 7x^3 - 1$ is used for all the variants of the LBA.

Algorithm	Time	Success rate
Algorithm 1 (LBA with backtracking)	48.68 hours	0%
Algorithm 2 (LBA with a dynamic set)	50.04 hours	0%
Algorithm 3 (Memory 1)	50.00 hours	0%
Algorithm 4 (Memory 2)	49.35 hours	3%

As we can see, Algorithm 4 has the best performance in this set of parameters, but even then, it has only 3% success rate. To further test Algorithm 4, we ran another set of tests where we increase the length of random elements to $[L_1, L_2] = [20, 23]$ and increase the number of factors of the private key to $L = 20$. To give it a chance of success, we increase the size of the memory M to 40,000. The result is still 0% success rate.

5.2.5. *The effect of increasing the time-out.* Since it is possible that the time-out of 30 minutes for each test is not enough, we ran another set of tests where the time-out is 4 hours for each test. Algorithm 4 showed the most promise, so we chose it with the following parameters: the length of random elements is in the interval $[L_1, L_2] = [20, 23]$, the number of factors of the private key is $L = 20$ and the size of the memory M is 1000. The polynomial used is $x^9 - 7x^3 - 1$ with Hirsch length 14. Due to the long time-out, we performed only 50 tests. We still get 0% success rate.

Based on the above experimental results, we conclude that polycyclic groups of high Hirsch lengths are resistant to the length-based attack.

5.2.6. *Additional experimental results concerning Algorithm 2.* These are some additional experimental results conducted with a time-out of 1 hour for each test. The polynomials used are f and $h(G)$ is the Hirsch length of the generated polycyclic group. The size of Alice's and Bob's public sets are N_1, N_2 respectively. Each random element a_i or b_i has length in $[L_1, L_2]$ and Alice's private key is the product of $L = 5$ random elements in Alice's public set. The success rate of a batch of 100 tests is recorded.

Polynomial	$h(G)$	$N_1 = N_2 = 5$		$N_1 = N_2 = 20$
		[5,8]	[15,18]	[10,13]
$x - 1$	1	98%		98%
$x^2 - x - 1$	3	98%	96%	100%
$x^3 - x - 1$	4	95%		100%
$x^5 - x^3 - 1$	7			35%
$x^7 - x^3 - 1$	10			8%
$x^9 - 7x^3 - 1$	14			5%
$x^{11} - x^3 - 1$	16	59%	53%	5%

REFERENCES

- [1] I. Anshel, M. Anshel, and D. Goldfeld. An algebraic method for public-key cryptography. *Math. Res. Lett.*, 6:287–291, 1999.
- [2] M. Anshel and D. Kahrobaei. Decision and search in non-abelian Cramer-Shoup public key cryptosystem. *Groups, Complexity, Cryptology*, 1:217–225, 2009.
- [3] B. Eick and D. Kahrobaei. Polycyclic groups: a new platform for cryptography, preprint arxiv: math.gr/0411077. Technical report, 2004.
- [4] B. Eick and W. Nickel. *Polycyclic: Computation with polycyclic groups, a GAP 4 package*, <http://www.gap-system.org/Packages/polycyclic.html>.
- [5] D. Garber, S. Kaplan, M. Teicher, B. Tsaban, and U. Vishne. Probabilistic solutions of equations in the braid group. *Adv. in App. Math.* 35, pages 323–334, 2005.
- [6] D. Garber, S. Kaplan, M. Teicher, B. Tsaban, and U. Vishne. Length-based conjugacy search in the braid group. *Contemp. Math.* 418, pages 75–87, 2006.
- [7] M. Hock and B. Tsaban. Solving random equations in Garside groups using length functions. *Combinatorial and Geometric Group Theory*, pages 149–169, 2010.
- [8] D. F. Holt, B. Eick, and E. A. O'Brien. *Handbook of computational group theory*. Chapman & Hall CRC, 2005.
- [9] J. Hughes and A. Tannenbaum. Length-based attacks for certain group based encryption rewriting systems. *Workshop SECI02 Securite de la Communication sur Internet*, 2002.
- [10] D. Kahrobaei and B. Khan. A non-commutative generalization of El-Gamal key exchange using polycyclic groups. *Proceedings of the Global Telecommunications Conference*, 4(2):1–5, 2006.

- [11] D. Kahrobaei and C. Koupparis. Non-commutative digital signatures using non-commutative groups. *Groups, Complexity, Cryptology*, 4:377–384, 2012.
- [12] K. H. Ko, S. J. Lee, J. H. Cheon, J. W. Han, J. Kang, and C. Park. New public-key cryptosystem using braid groups. *Advances in cryptology, CRYPTO 2000 (Santa Barbara, CA), LNCS, vol. 1880*, pages 166–183, 2000.
- [13] A. Myasnikov, V. Shpilrain, and A. Ushakov. *Non-commutative Cryptography and Complexity of Group-theoretic Problems*. American Mathematical Society, 2011.
- [14] A. D. Myasnikov and A. Ushakov. Length-based attack and braid groups: Cryptanalysis of Anshel-Anshel-Goldfeld key exchange protocol. *PKC 2007, LNCS 4450*, pages 76–88, 2007.
- [15] D. Ruinskiy, A. Shamir, and B. Tsaban. Length-based cryptanalysis: the case of Thompson’s group. *Journal of Mathematical Cryptology 1*, pages 359–372, 2007.
- [16] V. Shpilrain and A. Ushakov. Thompson’s group and public key cryptography. *ACNS*, pages 151–164, 2005.

DAVID GARBER, DEPARTMENT OF APPLIED MATHEMATICS, FACULTY OF SCIENCES, HOLON INSTITUTE OF TECHNOLOGY, 52 GOLOMB ST., PO BOX 305, 58102 HOLON, ISRAEL
E-mail address: `garber@hit.ac.il`

DELARAM KAHROBAEI, CUNY GRADUATE CENTER, PHD PROGRAM IN COMPUTER SCIENCE AND NYCCT, MATHEMATICS DEPARTMENT, CITY UNIVERSITY OF NEW YORK
E-mail address: `dkahrobaei@gc.cuny.edu`

HA T. LAM, DEPARTMENT OF MATHEMATICS, CUNY GRADUATE CENTER, CITY UNIVERSITY OF NEW YORK
E-mail address: `hlam@gc.cuny.edu`